```
 *
 * The first time C_FILTER is called Level=0, and then will be incremented
 * each time it is called for the same connection.
 *
 * Flags give the flags of the user (MGPBSXLE) in a bit field. as defined
 * in the INIT.SRV user's mask. (0x80=Modem, 0x40=Guest, etc...)
 *
 * If New=1, then this is the first connection of the user on the BBS.
 * Record is the record number in the INF.SYS file.
 *
 * All other arguments are the words sent by the user
 * (password for instance).
 *
 * The number of arguments is variable and depends of the number of words
 * in the answer of the user.
 *
 */


/*
 * This is only a little example to test the system. It will be called
 * four times and will give the list of arguments.
 *
 * The fourth time, the hand will be given back to the BBS.
 */

main(int ac, char **av)
{
  int i;
  int level = atoi(av[2]);            /* Get level from argument list  */
                                      /* and transform it to integer   */

  if (level == 0) {                   /* Is level equal to 0 ?         */
    printf("Connection line :\n");    /* This is the first call        */
    for (i = 0 ; i < ac ; i++)        /* List line arguments           */
    printf("%s ", av[i]);
    putchar('\n');
    return(1);                        /* C_FILTER must be called again */
  }
  else {
    printf("Following line  :\n");    /* These are other lines         */
    for (i = 0 ; i < ac ; i++)        /* List line arguments           */
      printf("%s ", av[i]);
    putchar('\n');
    if (level == 4)                   /* Is it the last time ?         */
      return(0);                      /* Yes, go on BBS                */
    else
      return(1);                      /* No, call once more C_FILTER   */
  }
}
```

APPENDIX -13- MESSAGE FILTERING.

FBB software allows filtering messages. Filtering is not done by the BBS software but by external programs developped by users.

may be interactive and allows to incorporate some features like dedicated information for predifined callsigns, password filtering, etc...

I did not develop such programs, but this is an open door to many applications.

The M_FILTER program must be found by the PATH of MsDos. Its extension can be COM or EXE, and it must be little and fast as multitasking is stopped during the activity of this program. If this program is not found, it will not be called until the BBS is rebooted.

The message filter is called (if found) each time a message is ready to be recorded (when Ctrl Z or /EX is received). The decision to validate or not the message is function of the exit value of the M_FILTER program.

The M_FILTER program (if found) is called with some arguments including a level number. This number is incremented each time the program is called in the same connection session. The first time the level number will be 0.

The line arguments given to the M_FILTER program are :

- File name including the text of the message.
- Type of the message (P, B, T).
- Sender.
- "To" field.
- Record number of DIRMES.SYS file.

The M_FILTER program ends with an exit value. This value is very important and tells the BBS what to do :

```
 0 : Message is recorded.
 1 : Message is killed   (status = K).
 2 : Message is archived (status = A).
 3 : Message is held     (status = H).
```

```
/*
 * M_FILTER.C
 *
 * The message filter MUST be named M_FILTER (COM or EXE).
 *
 * This example only writes its call arguments in the TEST.MES file.
 *
 * It is called with 5 arguments :
 *   File name of the message.
 *   Type .
 *   Sender.
 *   To.
 *   Number of the record in the DIRMES.SYS file.
 *
 * If it returns 0 : The message is accepted.
 *               1 : The message is killed (status K).
 *               2 : The message is archived (status A).
 *
 */

#include <stdio.h>
```

```
main(argc, argv)
int  argc;
char **argv;
{
  int  i;
  FILE * fptr = fopen("TEST.MES", "at");

  for (i = 0 ; i < argc ; fprintf(fptr, "%s ", argv[i++]));
  fputc('\n', fptr);

  fclose(fptr);

  return(0);
}
```

APPENDIX -14- PG programs development.

PG programs are in the PG subdirectory. They are little programs allowing interactivity with the user.

COM or EXE programs can be called.

PG programs must be little as the amount of memory is limited and fast because the multitasking is stopped during its activity.

To run a PG program and start a session, the user must type the command PG followed by the name of the program. The PG command alone gives the content of the PG subdirectory. The PG program is particularly developped for FBB software but can be an interface to a standard program.

Each time a complete line (up to the return character) is received, the PG program is called with some arguments including a level number. This number is incremented each time the program is called in the same PG session. The first time the level number will be 0.

The line arguments given to the PG program are :

- Callsign (format as F6FBB-8).
- Level number (0 is the first time, up to 99).
- Flags of the user (binary number as user's mask of INIT.SRV).
- Record number of the user in INF.SYS.
- Received data (each word is a new argument).

The PG program ends with an exit value. This value is very important and tells the BBS what to do :

0 : end of session and return to the BBS menu.
1 : the program will be called again and the level number is incremented.
2 : the user will be disconnected.
3 : the receive datas will be sent as a BBS command and return to BBS.
4 : the receive datas will be sent as a BBS command, level incremented.
5 : the program will be called again, but the level is not incremented.

The datas sent by the PG program to the standard output will be sent to the user. This allows a real interactivity between the user and the PG program.

Here is an example of a little program :

```
/*
 * TST_PG.C
 *
 * Little test program of "PG" command for FBB BBS software.
 *
 * (C) F6FBB 1991.
 *
 * FBB software 5.14 and upper.
 *
 *
 * This program echoes to the user what he types
 * or executes a BBS command preceeded by "CMD"
 * until "BYE" is received
 */

#include <stdio.h>

main(int argc, char **argv)
{
  int  i;
  int  level = atoi(argv[2]);        /* Get level from argument list   */

                                     /* and transform it to integer    */
  if (level == 0) {                  /* Is level equal to 0 ?          */
                                     /* This is the first call         */
    printf("Hello %s, type BYE when you want to stop !\n", argv[1]);
    return(1);                       /* program will be called again   */
  }
  else {
    strupr(argv[5]);                 /* Capitalise the first word      */
    if (strcmp(argv[5], "BYE") == 0) {       /* is BYE received ?      */
      printf("Ok, bye-bye\n");
      return(0);                     /* Yes, go on BBS                 */
    }
    else if (strcmp(argv[5], "CMD") == 0) {  /* is CMD received ?      */
      for (i = 6 ; i < argc ; i++)   /* List line arguments            */
      printf("%s ", argv[i]);        /* sent by user                   */
      putchar('\n');
      for (i = 6 ; i < argc ; i++)   /* List line arguments            */
        printf("%s ", argv[i]);      /* sent by user                   */
      putchar('\n');
      return(4);                     /* Yes, send command              */
    }
    else {
      printf("You told me : ");      /* These are other lines          */
      for (i = 5 ; i < argc ; i++)   /* List line arguments            */
      printf("%s ", argv[i]);        /* sent by user                   */
      putchar('\n');
      return(1);                     /* No, call again program         */
    }
  }
}
```

APPENDIX -15- UNPROTO LISTS.

FBB software allows sending unproto lists of message. This is validated separately on each port (letter L in PORT.SYS). Unproto address is FBB with the following header :

```
fm F6FBB-1 to FBB ctl UI
```

An unproto list line is sent on every validated port each time a message is recorded. The line is in the form :

```
12345 B 2053 TEST@ALL F6FBB 920325 This is the subject
```

If a message number is missing or does not exist the line will be :

```
12346 #
```

This allows a system listening to the UI packets on a frequency to create a list identical to the one of the BBS, and then the user will not have to connect the BBS to know the list of messages and bulletins.

A control can be done on the number of the messages to check if a line is missing.

If the remote system receives a new line, and a line is missing, it only has to send an unproto frame addressed to the BBS callsign like this :

```
fm FC1EBN-3 to F6FBB-1 ctl UI
? 00002EE00E
```

This will be taken in account only if the user has the U flag validated (EU command).

If the user has not his flag validated in the BBS, he will receive a line like :

```
fm F6FBB-1 to FBB ctl UI
12200 / callsign
```

In this case, the the remote software MUST stop asking unprotoes.

The first 8 digits are the hexadecimal number of the requested start of the list (here 00002EE0 -> 12000) and the last two digits are the sum of the four bytes anded with FF (0E).

The BBS will then starts sending lines from the requested number up to the last message number.

If the number requested seems to be too far from the current line, the BBS can reajust the request of "callsign" while sending :

```
fm F6FBB-1 to FBB ctl UI
12200 ! CALLSIGN
12201 B 2040    TEST@FRA    F6FBB  920325 This is a bulletin
12202 #
12203 P 206    F6ABJ@F6ABJ  F6FBB  920325 Hello Remy.
etc...
```

and then starts sending lines from 12201. The remote system must change its base number to 12201.

If the number requested is greater than the last message received in the BBS, the BBS will send a line like :

12300 !!

This indicates that the list in the remote system is up to date. The last received message in the BBS is 12300.

The remote system can also connect the BBS and ask for messages in binary compressed mode using the following sequence :

```
     BBS                                    Remote system
-------------------------------------------------------------------------
                                            Connection request
Connection.

[FBB-5.14-ABFHM$]
Welcome in Toulouse, Gerard.
F6FBB BBS>
                                            [TPK-1.80-$]
1>
                                            F< #Msg
Binary compressed message #Msg
is sent using format described
in appendix 7
1>
                                            Disconnect.
```

From TPK version 1.80 (packet communication program developped by FC1EBN), this protocole is implemented.

APPENDIX -16- EXTENSIONS TO THE YAPP PROTOCOLE.

These extensions are used in TPK 1.65a (and up) and FBB 5.14 and up.

HEADER extension.

The header now carries the DATE and TIME of the file being transmitted.
[SOH] [Len] [Filename] [NUL] [File Size] [NUL] [Date] [Time] [NUL]

The Date and Time are the values reported by DOS, coded in 4 hexadecimal digits and are sent in ASCII (8 characters).

The receiver has the choice of using either extended Yapp with checksum or normal Yapp.

The normal Yapp reply is RF, as before and the receiver can keep the date and time information .

The extended Yapp reply is : RT Receive_TPK and is coded : [ACK] [ACK]

If the receiver reply is RT the protocol used will be what I have called YappC for Yapp with checksum. When the sender gets this packet he MUST use YappC.

Data Packets extension.

If the receivers reply is RT the protocol used will be YappC. The checksum allows detection of packets corrupted along the link, particularly on the RS232 lines where there is no error control or correction (or it's very poor!)

Data packets : [STX] [Len] [Datas] [Checksum]

Checksum is the sum of all datas bytes anded with FF in 8 bits like Xmodem.

If the checksum is bad then the receiver must send a Cancel and enters CW state.

Crash Recovery.

A new field has been added to the resume reply to tell the sender if the receiver can use YappC or not. Resume is sent instead of RF (or RT).

Resume reply for Yapp: (as used before by TPK and FBB)

```
    [NAK] [Len] [R] [NUL] [Received Length] [NUL]
            I    I                I
            I    I                +-- in ASCII as in the header
            I    +-- as Resume !
            I
            +------- len of the following bytes
```

Resume reply for YappC:

```
    [NAK] [Len] [R] [NUL] [Received Length] [NUL] [C] [NUL]
                                                   I
                  Tells sender I can use YappC -----+
```

When the sender gets this packet then he must also use YappC.